Capsicum on Linux

David Drysdale
18 Aug 2014
Linux Security Summit
Capsicum

- Pragmatic application
- of object-capability principles
- to UNIX
- and Linux in particular
Capability-Based Security

- All object access needs a token: the *capability*
  - identifies the object
  - accompanying *rights* give allowed operations
Capability-Based Security

- All object access needs a token: the capability
  - identifies the object
  - accompanying rights give allowed operations

- Avoid object naming & ambient authority
  - Prevent confused deputy attacks
  - Acquire capabilities
    - by inheritance
    - by creation (with subset of rights)
    - by passing
Capability-Based Security

- All object access needs a token: the *capability*
  - identifies the object
  - accompanying *rights* give allowed operations

- Avoid object naming & ambient authority
  - Prevent confused deputy attacks
  - Acquire capabilities
    - by inheritance
    - by creation (with subset of rights)
    - by passing

- Note: completely different than POSIX.1e capabilities
Capsicum Principles

- POSIX File Descriptor Behaviour
Capsicum Principles

● POSIX File Descriptor Behaviour
● Hence Capsicum:
  ○ file descriptors as capabilities
  ○ with (new) fine-grained rights
    ■ policy co-located with code (ENOTCAPABLE)
Capsicum Principles

- POSIX File Descriptor Behaviour
- Hence Capsicum:
  - file descriptors as **capabilities**
  - with (new) fine-grained rights
    - policy co-located with code (**ENOTCAPABLE**)
  - **capability mode**:
    - prevent minting of new file descriptors
    - lock down global namespaces (**ECAPMODE**)

Example: tcpdump changes

```c
+   cap_rights_init(&rights, CAP_READ);
+   if (cap_rights_limit(fileno(pcap_file(pd)), &rights) < 0)
+     error("unable to limit pcap descriptor");

+   cap_rights_init(&rights, CAPSEEK, CAP_WRITE);
+   if (cap_rights_limit(fileno(pcap_dump_file(p), rights) < 0)
+     error("unable to limit dump descriptor");

+   if (cap_enter() < 0)
+     error("cap_enter: %s", pcap_strerror(errno));
status = pcap_loop(pd, cnt, callback, pcap_userdata);
```

But only with `-n` option (no reverse-DNS lookup)
Linux Implementation: Capabilities

- Rights associated with file descriptors
- Wrapper `struct file` object
Linux Implementation: Capabilities

- Rights associated with file descriptors
- Wrapper `struct file` object
- Check rights on all FD->`file` conversions
  - Annotate `fget()` operations with required rights
  - Altered error behaviour (`EBADF` or `ENOTCAPABLE`)
- Unwrap on all FD->`file` conversions
**fget** Annotation Example

```c
SYSCALL_DEFINE1(fchdir, unsigned int, fd)
{
    struct fd f = fdget_raw(fd);
    struct inode *inode;
    int error = -EBADF;

    if (!f.file)
        goto out;
...
```

```c
SYSCALL_DEFINE1(fchdir, unsigned int, fd)
{
    struct fd f = fdgetr_raw(fd, CAP_FCHDIR);
    struct inode *inode;
    int error = -EBADF;

    if (IS_ERR(f.file)) {
        error = PTR_ERR(f.file);
        goto out;
    }

    if (IS_ERR(f.file)) {
        error = PTR_ERR(f.file);
        goto out;
    }
}
...
```
Linux Implementation: Capabilities

- Rights associated with file descriptors
- Wrapper \texttt{struct file} object
- Check rights on all FD->\texttt{file} conversions
  - Annotate \texttt{fget()} operations with required rights
  - Altered error behaviour (\texttt{EBADF} or \texttt{ENOTCAPABLE})
- Unwrap on all FD->\texttt{file} conversions
- Wrap new FDs from existing FDs on install
Linux Implementation: Capabilities

- Rights associated with file descriptors
- Wrapper `struct file` object
- Check rights on all FD->`file` conversions
  - Annotate `fget()` operations with required rights
  - Altered error behaviour (**EBADF** or **ENOTCAPABLE**)
- Unwrap on all FD->`file` conversions
- Wrap new FDs from existing FDs on install
- Prevent non-relative `openat` (**O_BENEATH**)
Linux Implementation: Capability Mode

- Prevent syscalls that access global namespaces
  - Use seccomp-bpf
  - New ECAPMODE errno
Linux Implementation: Capability Mode

- Prevent syscalls that access global namespaces
  - Use seccomp-bpf
  - New **ECAPMODE** errno

- Wrinkles
  - Process-wide filter
  - Prevent non-relative filesystem access
  - Allow self-signal (**kill** / **tgkill**)?
Process Descriptors

- Manipulating sub-processes is useful
  - compartmentalize into sandboxed sub-processes
Process Descriptors

- Manipulating sub-processes is useful
  - compartmentalize into sandboxed sub-processes
- Add **process descriptors**
  - file descriptor wrapper for *pid_t*
  - *pdfork/pdkill/pdwait4*
Process Descriptors

- Manipulating sub-processes is useful
  - compartmentalize into sandboxed sub-processes
- Add **process descriptors**
  - file descriptor wrapper for `pid_t`
  - `pdfork/pdkill/pdwait4`
- Avoid perturbing rest of application
  - No `SIGCHLD` on exit
  - Not visible to `waitpid(-1,...)`
Capsicum Status

- Experimental in FreeBSD 9.x (2012)
- Included in FreeBSD 10.x (2014)
  - ~12 sandboxed utilities in tree
  - OpenSSH & Chromium out of tree
- Linux patchset proposed on LKML (2014)
  
  https://lkml.org/lkml/2014/7/25/426
  https://github.com/google/capsicum-linux
  https://github.com/google/capsicum-test